



# UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE  
United States Patent and Trademark Office  
Address: COMMISSIONER FOR PATENTS  
P.O. Box 1450  
Alexandria, Virginia 22313-1450  
www.uspto.gov

| APPLICATION NO. | FILING DATE | FIRST NAMED INVENTOR | ATTORNEY DOCKET NO. | CONFIRMATION NO. |
|-----------------|-------------|----------------------|---------------------|------------------|
| 09/818,130      | 03/27/2001  | Noam Livnat          |                     | 8644             |

7590

07/13/2004

Noam Livnat  
320 Waverley St., #1  
Menlo Park, CA 94025

|          |
|----------|
| EXAMINER |
|----------|

PHAN, TAM T

|          |              |
|----------|--------------|
| ART UNIT | PAPER NUMBER |
|----------|--------------|

2144

DATE MAILED: 07/13/2004

2

Please find below and/or attached an Office communication concerning this application or proceeding.

**RECEIVED**

**JUL 29 2004**

**Technology Center 2100**

## Office Action Summary

Application No.

09/818,130

Applicant(s)

LIVNAT, NOAM

Examiner

Tam (Jenny) Phan

Art Unit

2144

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

### Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

### Status

- 1) ☒ Responsive to communication(s) filed on 27 July 2001.
- 2a) ☐ This action is **FINAL**. 2b) ☒ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

### Disposition of Claims

- 4) ☒ Claim(s) 1-30 is/are pending in the application.
- 4a) Of the above claim(s) \_\_\_\_\_ is/are withdrawn from consideration.
- 5) ☐ Claim(s) \_\_\_\_\_ is/are allowed.
- 6) ☒ Claim(s) 1-30 is/are rejected.
- 7) ☐ Claim(s) \_\_\_\_\_ is/are objected to.
- 8) ☐ Claim(s) \_\_\_\_\_ are subject to restriction and/or election requirement.

RECEIVED

JUL 29 2004

Technology Center 2100

### Application Papers

- 9) ☐ The specification is objected to by the Examiner.
- 10) ☒ The drawing(s) filed on 27 March 2001 is/are: a) ☒ accepted or b) ☐ objected to by the Examiner.
- Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
- Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

### Priority under 35 U.S.C. § 119

- 12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☐ All b) ☐ Some \* c) ☐ None of:
- ☐ Certified copies of the priority documents have been received.
  - ☐ Certified copies of the priority documents have been received in Application No. \_\_\_\_\_.
  - ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

\* See the attached detailed Office action for a list of the certified copies not received.

### Attachment(s)

- 1) ☒ Notice of References Cited (PTO-892)
- 2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
- 3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08)  
Paper No(s)/Mail Date \_\_\_\_\_
- 4) ☐ Interview Summary (PTO-413)  
Paper No(s)/Mail Date. \_\_\_\_\_
- 5) ☐ Notice of Informal Patent Application (PTO-152)
- 6) ☐ Other: \_\_\_\_\_

**DETAILED ACTION**

1. This application has been examined. Claims 1-30 presented for examination.

***Priority***

2. This application claims benefit of the provisional application 60/193,753 (03/31/2000).
3. The effective filing date for the subject matter defined in the pending claims, which has support in Provisional Application No. 60/193,753 is 03/31/2000. Any new subject matter defined in the claims not previously disclosed in Provisional Application No. 60/193,753, is entitled to the effective filing date of 03/27/2001.

***Claim Rejections - 35 USC § 102***

4. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(e) the invention was described in a patent granted on an application for patent by another filed in the United States before the invention thereof by the applicant for patent, or on an international application by another who has fulfilled the requirements of paragraphs (1), (2), and (4) of section 371(c) of this title before the invention thereof by the applicant for patent.

5. Claims 1-3, 9, 11-18, 24, and 26-30 rejected under 35 U.S.C. 102(e) as being anticipated by Shoroff et al. (U.S. Patent Number 6,381,602), hereinafter referred to as Shoroff.
6. Regarding claim 1, Shoroff disclosed a system for secure storage of information and controlled grant of access to said information to clients on a network, said system comprising: a server; a client computer coupled to said server via said network; a datastore configured to store said information; and an access controller coupled between said server and said datastore, wherein said access controller is adapted to function as an application server and provide a data representation of said information to said client by way of said server and said network as a

Art Unit: 2144

function of: (1) a request from said client sent by way of said network and said server; and (2) predetermined criteria; wherein said data representation is transient in said server (Abstract, Figures 2-4, column 3 lines 11-53, column 6 line 64-column 7 line 6, column 7 lines 15-24, column 9 lines 22-54).

7. Regarding claim 2, Shoroff disclosed a system wherein said network includes the Internet and World Wide Web (column 6 lines 38-44).

8. Regarding claim 3, Shoroff disclosed a system wherein said network includes a telephone network and said system includes a telephone coupled to said access controller via said telephone network [networking environments] (column 6 lines 31-44, line 64-14). Note: telephone network is well known in the networking environments.

9. Regarding claim 9, Shoroff disclosed a system wherein said request from said client includes a client identification and an information identification (column 9 lines 22-54, column 10 lines 9-31).

10. Regarding claim 11, Shoroff disclosed a system wherein said predetermined criteria may be different for different client types (Figure 4 sign 102', Figure 7 sign 120, 122, column 3 lines 11-53, column 10 lines 9-31).

11. Regarding claim 12, Shoroff disclosed a system wherein said information includes a plurality of content items and said access controller provides to a graphical user interface of said client computer a client selectable content list, indicating content items for which said data representations can be provided to said client, wherein said client may generate said request by selecting a desired content item from said content list (Figures 3-5, column 3 lines 31-53, column 9 lines 22-38, column 11 lines 29-43).

12. Regarding claim 13, Shoroff disclosed a system wherein a graphical user interface of a client computer includes mechanisms to facilitate said client generating said request by entering a URL, entering a content item identification, performing a text search, or manipulating a directory tree (column 3 lines 11-30, column 7 lines 44-56, column 9 lines 22-54).

13. Regarding claim 14, Shoroff disclosed a system wherein said criteria include criteria for verifying that said client is entitled to be granted access to said information, said criteria for verifying including an identification of said user (Figure 7 sign 134, column 1 lines 39-49, column 9 lines 21-38).

14. Regarding claim 15, Shoroff disclosed a system wherein said data representation is provided as a further function of history [cache] and profile information associated with said client (Figure 6 sign 108, Figure 7 sign 134, column 1 lines 39-49, column 2 lines 56-58, column 3 lines 12-30, lines 42-53, column 9 lines 21-38, column 10 lines 10-31, column 11 line 46-column 12 line 19).

15. Regarding claims 16-18, 24, and 26-30, the method corresponds directly to the system of claims 1-2, 9, and 11-15, and thus these claims are rejected using the same rationale.

16. Since all the limitations of the claimed invention were disclosed by Shoroff, claims 1-3, 9, 11-18, 24, and 26-30 are rejected.

17. Claims 1-2, 9-17, and 24-30 rejected under 35 U.S.C. 102(e) as being anticipated by Mehring et al. (U.S. Patent Number 6,609,115), hereinafter referred to as Mehring.

18. Regarding claim 1, Mehring disclosed a system for secure storage of information and controlled grant of access to said information to clients on a network, said system comprising: a

Art Unit: 2144

server; a client computer coupled to said server via said network; a datastore configured to store said information; and an access controller coupled between said server and said datastore, wherein said access controller is adapted to function as an application server and provide a data representation of said information to said client by way of said server and said network as a function of: (1) a request from said client sent by way of said network and said server; and (2) predetermined criteria; wherein said data representation is transient in said server (Abstract, Figures 4-8, column 2 lines 31-42, column 2 line 61-column 3 line 10, column 8 lines 9-37, column 9 lines 9-52, column 10 lines 4-31).

19. Regarding claim 2, Mehring disclosed a system wherein said network includes the Internet and World Wide Web (Figures 4-5, column 3 lines 11-21, column 9 lines 9-17).

20. Regarding claim 9, Mehring disclosed a system wherein said request from said client includes a client identification and an information identification (column 2 lines 31-42, column 3 lines 1-10, column 8 lines 9-37, column 9 lines 28-52).

21. Regarding claim 10, Mehring disclosed a system wherein said clients are typed and said data representation is provided to said client as a further function of a client type (column 13 lines 39-61, column 14 lines 44-63).

22. Regarding claim 11, Mehring disclosed a system wherein said predetermined criteria may be different for different client types (column 13 lines 38-61, column 14 lines 44-63).

23. Regarding claim 12, Mehring disclosed a system wherein said information includes a plurality of content items and said access controller provides to a graphical user interface of said client computer a client selectable content list, indicating content items for which said data representations can be provided to said client, wherein said client may generate said request by

Art Unit: 2144

selecting a desired content item from said content list (Figures 6 and 8, column 7 lines 1-28, column 9 lines 8-17, column 10 lines 4-31).

24. Regarding claim 13, Mehring disclosed a system wherein a graphical user interface of a client computer includes mechanisms to facilitate said client generating said request by entering a URL, entering a content item identification, performing a text search, or manipulating a directory tree (Figures 6 and 8, column 9 lines 9-17).

25. Regarding claim 14, Mehring disclosed a system wherein said criteria include criteria for verifying that said client is entitled to be granted access to said information, said criteria for verifying including an identification of said user (Figures 6 and 8, column 2 lines 31-42, column 8 lines 9-37).

26. Regarding claim 15, Mehring disclosed a system wherein said data representation is provided as a further function of history and profile information associated with said client (column 11 lines 1-24).

27. Regarding claims 16-17 and 24-30, the method corresponds directly to the system of claims 1-2 and 9-15, and thus these claims are rejected using the same rationale.

28. Since all the limitations of the claimed invention were disclosed by Mehring, claims 1-2 and 9-17, 24-30 are rejected.

### ***Claim Rejections - 35 USC § 103***

29. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

30. Claims 4-8, 10, 19-23, and 25 are rejected under 35 U.S.C. 103(a) as being unpatentable over Shoroff et al. (U.S. Patent Number 6,381,602), hereinafter referred to as Shoroff, as applied to above in view of Ballard (U.S Patent Number 6,182,050).

31. Regarding claim 4, Shoroff disclosed a system for secure storage of information and controlled grant of access to said information to clients on a network, said system comprising: a server; a client computer coupled to said server via said network; a datastore configured to store said information; and an access controller coupled between said server and said datastore, wherein said access controller is adapted to function as an application server and provide a data representation of said information to said client by way of said server and said network as a function of: (1) a request from said client sent by way of said network and said server; and (2) predetermined criteria; wherein said data representation is transient in said server (Abstract, Figures 2-4, column 3 lines 11-53, column 6 line 64-column 7 line 6, column 7 lines 15-24, column 9 lines 22-54).

32. Shoroff taught the invention substantially as claimed. However, Shoroff did not expressly teach a system wherein said predetermined criteria define a time window for which said information is available for access.

33. Shoroff suggested exploration of art and/or provided a reason to modify the system with the time window access feature (column 12 lines 1-19).

34. Ballard disclosed a system wherein said predetermined criteria define a time window for which said information is available for access.

35. It would have been obvious to one of ordinary skill in the art at the time of the invention was made to modify the system of Shoroff with the teachings of Ballard to include the time



Art Unit: 2144

window access feature in order to monitor the access information (Ballard, column 7 lines 36-55) since it is beneficial to allow the distribution system to be automated over a public networks (Ballard, column 8 lines 7-21). In addition, if the information is available for access relates to broadcast or advertisement, the owner of the information might only want to pay to have their information distributed to a requisite number of end users within a specific period of time (Ballard, column 10 lines 54-67).

36. Regarding claim 5, Ballard disclosed a system wherein said criteria includes a start date, wherein said start date defines when said information is made available for access (column 7 lines 41-49, column 10 lines 54-67, column 17 lines 36-61, column 18 lines 33-50).

37. Regarding claim 6, Ballard disclosed a system wherein said criteria includes a period of duration of access, wherein said period of duration of access commences upon said information being accessed by said client (column 7 lines 41-49, column 10 lines 54-67, column 17 lines 36-61, column 18 lines 33-50).

38. Regarding claim 7, Ballard disclosed a system wherein said criteria includes an end date, wherein said end date defines when said information ceases to be available for access (column 7 lines 41-49, column 10 lines 54-67, column 17 lines 36-61, column 18 lines 33-50).

39. Regarding claim 8, Ballard disclosed a system wherein said criteria includes a start date and a start time, wherein said start date and start time define when said information is made available for access, and further includes an end date and an end time, wherein said end date and end time define when said information ceases to be available for access (column 7 lines 41-49, column 10 lines 54-67, column 17 lines 36-61, column 18 lines 33-50).

Art Unit: 2144

40. Regarding claim 10, Ballard disclosed a system wherein said clients are typed and said data representation is provided to said client as a further function of a client type (column 9 lines 22-67).

41. Regarding claims 19-23, and 25, the method corresponds directly to the system of claims 4-8, and 10, and thus these claims are rejected using the same rationale.

42. Since all the limitations of the claimed invention were disclosed by the combination of Shoroff and Ballard, claims 4-8, 10, 19-23, and 25 are rejected.

### ***Conclusion***

43. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure. Refer to the enclosed PTO-892 for details.

44. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Tam (Jenny) Phan whose telephone number is (703) 305-4665. The examiner can normally be reached on M-F 9:00-5:00.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, William Cuchlinski can be reached on 703-308-3873. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the receptionist whose telephone number is (703) 305-3900.


Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR

Art Unit: 2144

system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

William Cuchlinski  
SPE  
Art Unit 2144  
703-308-3873

tp  
July 6, 2004



WILLIAM A. CUCHLINSKI, JR.  
SUPERVISORY PATENT EXAMINER  
TECHNOLOGY CENTER 3600

|                                   |                                       |  |             |
|-----------------------------------|---------------------------------------|--|-------------|
| <b>Notice of References Cited</b> | Application/Control No.<br>09/818,130 | Applicant(s)/Patent Under<br>Reexamination<br>LIVNAT, NOAM |             |
|                                   | Examiner<br>Tam (Jenny) Phan          | Art Unit<br>2144   | Page 1 of 1 |

#### U.S. PATENT DOCUMENTS

| * |   | Document Number<br>Country Code-Number-Kind Code | Date<br>MM-YYYY | Name                 | Classification |
|---|---|--|-----------------|----------------------|----------------|
|   | A | US-6,381,602                                     | 04-2002         | Shoroff et al.       | 707/9          |
|   | B | US-6,182,050                                     | 01-2001         | Ballard, Clinton L.  | 705/14         |
|   | C | US-6,609,115                                     | 08-2003         | Mehring et al.       | 705/51         |
|   | D | US-5,787,428                                     | 07-1998         | Hart, Keith          | 707/9          |
|   | E | US-6,044,373                                     | 03-2000         | Gladney et al.       | 707/10         |
|   | F | US-6,466,983                                     | 10-2002         | Strazza, Steven Paul | 709/227        |
|   | G | US-6,704,787                                     | 03-2004         | Umbreit, Timothy F.  | 709/229        |
|   | H | US-6,662,181                                     | 12-2003         | Icken et al.         | 707/9          |
|   | I | US-  |                 |                      |                |
|   | J | US-  |                 |                      |                |
|   | K | US-  |                 |                      |                |
|   | L | US-  |                 |                      |                |
|   | M | US-  |                 |                      |                |

#### FOREIGN PATENT DOCUMENTS

| * |   | Document Number<br>Country Code-Number-Kind Code | Date<br>MM-YYYY | Country | Name | Classification |
|---|---|--|-----------------|---------|------|----------------|
|   | N |  |                 |         |      |                |
|   | O |  |                 |         |      |                |
|   | P |  |                 |         |      |                |
|   | Q |  |                 |         |      |                |
|   | R |  |                 |         |      |                |
|   | S |  |                 |         |      |                |
|   | T |  |                 |         |      |                |

#### NON-PATENT DOCUMENTS

| * |   | Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)   |
|---|---|---|
|   | U | Bertino, E.; Jajodia, S.; Samarati, P.; "Supporting multiple access control policies in database systems" Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on , 6-8 May 1996 Pages:94 - 107  |
|   | V | Deng, P.; Kuo, C.; Kao, V.; "A dynamic access control model for object-oriented system" Security Technology, 1993. Security Technology, Proceedings. Institute of Electrical and Electronics Engineers 1993 Conference on , 13-15 Oct. 1993 Pages:159 - 163 |
|   | W |   |
|   | X |   |

\*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)  
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

## A Dynamic Access Control Model for Object-Oriented System

Peter Shaohua Deng\*, Chenyuan Kuo\*, Victor T. S. Kao\*\*

### ABSTRACT

In this paper we propose a Dynamic Access Control Model (DACM) to meet the access control requirements of an environment that frequently change and adjust their user access privilege and/or target object confidential level.

For DACM, we use object-orient technique in describing the whole model and also in implementing its access control mechanism called Dynamic Access Control System (DACS). For DACS, there are three server object modules within it to satisfy the request of access authorization check issued from any Client Object (CO) module. The three server objects of DACS are: (1) User\_Security\_Manager (USM) which dynamically maintains user privilege information stored in a persistent table called User\_Security\_Table (UST); (2) Target\_Object\_Manager (TOM) which dynamically maintains the access authorization rules stored in the persistent table called Target\_Object\_Table (TOT) for every target object; and (3) Access\_Control\_Manager (ACM) which is a higher level object module upon USM and TOM and performs a rule-based access authorization checking procedure for the Client Object. Also, two assistant tools called User Privilege List and Target Object Authorization List are provided as visual aids for the security controller of DACM to implement their access control policy.

A real case similar to DACM has been implemented in the Administration Information System of Central Police University, the performance shown in this case is quite good in the viewpoints of both end user and system developers/maintainers. We find that DACM technique can really be used to cope with the dynamically changing environment.

Finally, a discussion was made about several open questions and possible solutions for those DACM implementors. Since Environments are different for each DACM implementor, so that there are really some thing different in the consideration.

**KEYWORD:** ACCESS CONTROL, OBJECT-ORIENTED, SECURITY, RULE-BASED, ODMS

\* Instructor of department of information management, Central Police University

\*\* professor and chief of computer center, Central Police University

### 1. INTRODUCTION

Access Control in information system refers to use a mechanism to protect shared information or other resources among users via a mechanism[11, 14]. The access control mechanism can enforce the policies governing resource use. An access control model is a description of the whole protected system including the access control mechanism. Traditionally, there are several ways to describe an access control model, for example, access matrix, access list, capability list, protection domain, access domain, ...etc.; also there are a lot of access control technique developed, for example, lock-key mechanism, capability-based system, ring structure system, user classification by group, user classification by privilege level, ... etc[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14].

Since real world are so dynamic and changing that access control policies are changed frequently. For example, a user access privilege

may need to be changed because of his promotion or transferring department in a organization, on the other hand, the access authorization rules of a target object in an information system[2, 4, 12, 13] may change because of some real world situation.

To satisfy the access control requirements of Information system to cope with the dynamically changing environment, we propose a new access control technique called Dynamic Access Control Model (DACM) which can be implemented in an object-oriented software system to provide a rule-based access control mechanism.

### II. DYNAMIC ACCESS CONTROL MODEL (DACM)

#### 1. DACM Modeling

In DACM environment, we suppose that there is an object-oriented (OO) information system with a security controller who centralized performs all the information access control tasks. Figure 1 describes the whole system environment of DACM, where Dynamic Access Control System (DACS) provides an access control mechanism for all the object-oriented information system, and the Object Database Management System (ODMS) manages all the data and methods of object modules of the information system.

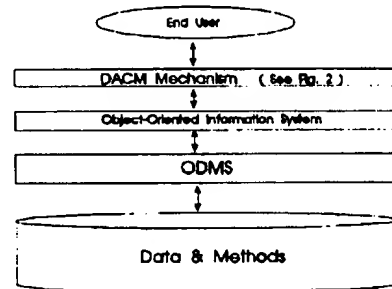


Figure 1. Dynamic access control model

#### 2. Components in DACS

Figure 2 is a detail look of Dynamic Access Control System (DACS), where we can see that as soon as an external object (called client object, CO) issues a request for an access authorization check by calling Access\_Control\_Manager (ACM) (like a server object), the ACM immediately perform a rule-based access authorization checking procedure according to the message passed from User\_security\_Manager (USM) and Target\_Object\_Manager (TOM), and then passes the message of checking result back to the Client Object (CO). The client object (CO) is a general term for any object which is external the DACS and wishes to has an access protection function surround it.

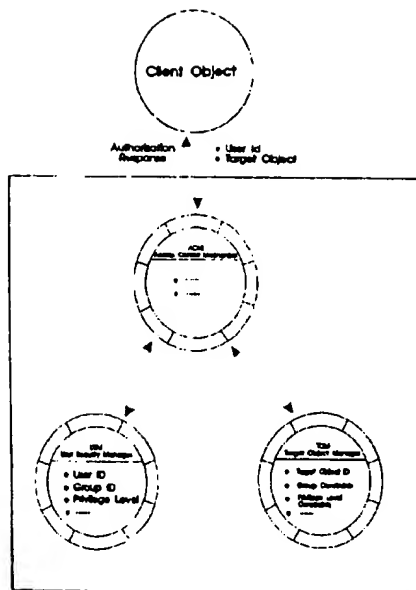


Figure 2. Internal objects and tables in DACS

### 3. Tables and Objects in DACS

#### 3.1. Tables

##### 3.1.1. User\_Security\_Table ( UST )

It is a persistent table[13] storing all the users' access privilege information, and that there is one entry for each user. The attributes in the table are:

| ATTRIBUTES       | REMARK         |
|------------------|----------------|
| .user_id         | an indexed key |
| .user_name/title |                |
| .group_code      | blank for all  |
| .privilege_level | value is 0-9   |

##### 3.1.2. Target\_Object\_Table ( TOT )

It is a persistent table storing all the access authorization rules for target objects, and that there can be one or multiple entries (rules) for each target object. If there is no entry in this table for a specific target object, it means that this target object can be accessed by all of the users in the system.

| ATTRIBUTES                          | REMARK         |
|-------------------------------------|----------------|
| .target_object_id                   | an indexed key |
| .target_object_name/title           |                |
| .authorized_user_id                 | blank for all  |
| .authorized_group_code              | blank for all  |
| .minimal_accessible_privilege_level | value is 0-9   |

#### 3.2. Objects

We will use C++ like semantic here to describe all the objects, and suppose that USM, TOM and ACM represent one of the objects of classes User\_Security\_Manager, Target\_Object\_Manager and

Access\_Control\_Manager separately. The following is a description of those data, methods (i.e. procedures in C++) and/or algorithms for each object used in DACS.

##### 3.2.1. User\_Security\_Manager ( USM )

###### . Data

has several transient data used to map with those persistent data stored in UST.

###### . Methods

- get\_privilege(uid\_a, gcode\_a, plevel\_a, rcode\_a)  
 . function - to retrieve user group code and privilege level information from UST for ACM.  
 . parameter -  
 . uid\_a - inputted user id passed from ACM  
 . gcode\_u - returned group code retrieved from UST  
 . plevel\_a - returned privilege level retrieved from UST  
 . rcode\_a - returned code, 0: OK, 1: INVALID USER-ID

- check\_password(uid\_a, pwd\_a, rcode\_a)  
 . function - to perform login user password check for the login handling object.

- ( other methods )  
 . function - to provide insertion, modification, deletion and inquiry function to maintain User\_Security\_Table (UST).

##### 3.2.2. Target\_Object\_Manager ( TOM )

###### . Data

has several transient data used to map with those persistent data stored in TOT.

###### . Methods

- get\_first(tid\_b, uid\_b, gcode\_b, plevel\_b, rcode\_b)  
 . function - to retrieve the first access authorization rule entry from table TOT for the input target object id.  
 . parameter -  
 . tid\_b - inputted target object id passed from ACM  
 . uid\_b - returned user id retrieved from TOT  
 . gcode\_b - returned group code retrieved from TOT  
 . plevel\_b - returned privilege level retrieved from TOT  
 . rcode\_b - returned code, 0: OK, 1: NOT FOUND

- get\_next(uid\_b, gcode\_b, plevel\_b, rcode\_b)  
 . function - to retrieve the next access authorization rule entry from table TOT for the input target object id.

- ( other methods )  
 . function - to provide insertion, modification, deletion and inquiry function to maintain Target-Object\_Table (TOT).

##### 3.2.3. Access\_Control\_Manager ( ACM )

###### . Data

has several transient data used to perform the rule-based access authorization checking procedure.

###### . Methods

- access\_control(uid, tid, rcode)  
 . function - to provide a rule-based algorithm to make the decision

of access authorization according to the input parameters and the message from USM and TOM.

. parameter  
 . tid\_h - inputted target object id from the client object  
 . uid\_b - inputted user id from the client object  
 . rcode\_b - returned code, 0: ACCESS ACCEPTABLE  
 1: ACCESS DENIED  
 2: INVALID USER-ID

. Algorithm for Procedure Access\_Control (In C++ sementle)

uid and tid is passed from the client object ;

CLASS user\_security\_manager ; // declare a class  
 CLASS target\_object\_manager ;

PROCEDURE access-control ( uid, tid, rcode )

```
{
  user_security_manager usm ; // usm is an object of
                             ser_security_manager
  target_object_manager tom ; // tom is an object of
                             target_object_manager
  usm.get_privilege(uid, gcode_a, // get user privilege information
                  plevel_a, rcode_a) ; // from table UST.
  IF ( rcode_a != 0 ) // test whether the return code is OK ?
    rcode = 2 ;
  ELSE // the user id is valid.
  {
    tom.get_first(tid, uid_b, gcode_b, // get the first entry from table TOT
                plevel_b, rcode_b) ; // for the input target object -tid.
    IF ( rcode_b != 0 ) // are there any rules for tid_b ?
      rcode = 0 ; // all users can access this target object.
    ELSE
    {
      rcode = 1 ;
      WHILE ( rcode_b == 0 and rcode == 1 ) // loop until end of entry
        or access is authorized.
      {
        IF ( uid_b is not blank ) // is this access authorization for a
                                specific user or a class of users ?
        {
          IF ( uid_a == uid_b )
            rcode = 0 ; // access authorized.
          ELSE
          {
            IF ( gcode_a is blank OR // If user has no group code const-
                gcode_b is blank ) // rant OR If target object has no
                                // group code constraint, THEN ...
            {
              IF ( plevel_a >= plevel_b ) // is the user privilege level
                rcode = 0 ; // high enough ?
            }
            ELSE
            {
              IF ( gcode_a == gcode_b )
                IF ( plevel_a >= plevel_b )
                  rcode = 0 ;
            }
          }
        }
        tom.get_next(tid, uid_b, gcode_b, // get the next entry from
                plevel_b, rcode_b) ; // table TOT for the input
                                // target object.
      }
    }
  }
}
```

pass the return code (rcode) back to the client object ;

### III. Tools for DACM

Besides traditional access matrix, capability list and access list, two other lists are proposed as visual aids for the security controller of DACM to facilitate his routine job.

#### 1. User Security List

It is generated from user security table and sorted by USFR ID, the format and illustration is shown in Table 1. For example in Table 1, USER-01 is authorized to access those objects which can be used by group G001 and required minimal privilege level is higher than 4, however, USER-04 is authorized to access all the objects with only required minimal privilege level not higher than 6.

#### 2. Target Object Authorization List

It is generated from target object table and sorted by TARGET OBJECT ID, the format and illustration is shown in Table 2. For example in Table 2, entry 1 indicates that USER-03 is authorized to access OBJ-01. Also, entry 2 indicates that those user(s) with group code G001 or blank and privilege level not lower than 5 are authorized to access object OBJ-01. And entry 6 represents that all those users with privilege level not lower than 6 are authorized to access object OBJ-06.

According to the illustration in Table 1 and Table 2, we can get an access matrix as follows, note that OBJ-07 is a target object with no rule in the Target\_Object\_Table, so it means that OBJ-07 can be accessed by all of the users in the system.

### IV. Example of DACM

A real case similar to DACM has been implemented in the Administration Information System of Central Police University. The performance is quite good in the viewpoints of both end users and system developers. To the security controller's viewpoints, it is convenient to manage the dynamic changing environment by tuning the user security table (UST) when a new user is added or when the access privilege domain of an existed user should be modified, or tuning the target object table (TOT) when several access protection rules should be put on an object or when the confidential level of an existed object needs to be upgraded or downgraded to meet the real world situation. Also, the routine access control job of the DACM security controller is facilitated via using convenient interactive end-user interface to maintain UST and TOT, and via using flexible visual aids, i.e., user security list, target object authorization list, capability list, access list and/or access matrix.

On the other hand, to the system developers and maintainers viewpoints, DACM technique is quite simple and is clear to implement into the information system. And also, since we are using the object-oriented technique in DACM, so that the Dynamic Access Control System (DACS) can be treated as an access control server, whenever a new system module is incorporated into the existed information system, it can also share the DACS protection function only by embedding the calling access\_control\_manager (ACM) instruction in it as playing a role of the Client Object (CO) to the ACM.

Table 1. User Security List

| SEQ No. | USER ID | USER TITLE     | (USER'S) GROUP CODE | (USER'S) PRIVILEGE LEVEL |
|---------|---------|----------------|---------------------|--------------------------|
| 1       | USER-01 | Peter Bush     | G0001               | 4                        |
| 2       | USER-02 | Robert Smith   | G0002               | 8                        |
| 3       | USER-03 | Mary Rogers    | G0003               | 5                        |
| 4       | USER-04 | Reaver Ken     | (blank)             | 6                        |
| 5       | USER-05 | Sophia Clinton | (blank)             | 9                        |

Table 2. Target Object Authorization List

| SEQ No. | TARGET OBJECT ID | AUTHORIZED USER ID | AUTHORIZED GROUP CODE | MINIMAL PRIVILEGE LEVEL |
|---------|------------------|--------------------|-----------------------|-------------------------|
| 1       | OBJ-01           | USER-03            | n/a                   | n/a                     |
| 2       | OBJ-02           | (blank)            | G001                  | 5                       |
| 3       | OBJ-03           | (blank)            | G001                  | 4                       |
| 4       | OBJ-04           | (blank)            | G001                  | 6                       |
| 5       | OBJ-05           | (blank)            | G003                  | 4                       |
| 6       | OBJ-06           | (blank)            | (blank)               | 3                       |
| 7       | OBJ-07           | (blank)            | (blank)               | 7                       |

\* n/a - not available

Table 3. Access Matrix

|         | OBJ-01 | OBJ-02 | OBJ-03 | OBJ-04 | OBJ-05 | OBJ-06 | OBJ-07 |
|---------|--------|--------|--------|--------|--------|--------|--------|
| USER-01 | no     | yes    | no     | no     | yes    | no     | yes    |
| USER-02 | yes    | yes    | yes    | no     | yes    | yes    | yes    |
| USER-03 | yes    | no     | no     | yes    | yes    | no     | yes    |
| USER-04 | yes    | yes    | yes    | yes    | yes    | no     | yes    |
| USER-05 | yes    | yes    | yes    | yes    | yes    | yes    | yes    |

#### V. Discussion

1. We can see from the real case in part IV that DACM technique is actually workable and can provide a quite good performance both in the viewpoints of end users, system developers and system maintainers.
2. Since this paper is a simplified description of DACM, so there are some areas need to be modified or taking into consideration for practical implementation viewpoints
  - (1). Group code can be multiple to meet the requirements of a multi-level organization. For example, someone can only access all the objects which belongs to those different sub-groups of a higher level group
  - (2). How to keep the user-id after end user logged in, that is, how can we let ACM to know the login user id? For example, we can keep the user id in a transient or persistent table is one of the way, or transfer it object-by-object is another way.
  - (3). How to naming the target object id's? For example, the target object id may represent a function, a table, or any other resources which needs to be protected.
  - (4). What if user need to classify different kind of access right for a specific target object? For example, read, write, modify, delete and/or execute ... etc. The possible solution could be: (i). naming different id's for the different kind of access right of a specific target object. (ii). add some additional attribute/field into target\_object\_table (TOT) to represent the availability of the different kind of access right.
  - (5). Convenient user interfaces for security controllers are required. For example, (i). interfaces to maintain user\_security\_table and target\_object\_table. (ii). interfaces to browse or print user security list, target object authorization list, capability list, access list and access matrix.

3. The similar concept of DACM seems can be implemented in an Object Database Management System as a common access control tool.

#### REFERENCES

- [1] Udo Keller, "Discretionary access controls in a high-performance object management system", Proceedings of Security and Privacy, 1991, pp255-275.
- [2] Sushil Jajodia, Boris Kogan, "Integrating an object-oriented data model with multilevel security", Proceedings of Security and Privacy, 1990, pp76-85.
- [3] Jian-Ke Jan, "Single-key access control control scheme in information protection system", Information Science vol. 51, no. 1, Jun 1990, pp1-11.
- [4] E. B. Fernandez, E. Gudes, H. Song, "Security model for object-oriented databases, Proceedings of Security and Privacy, 1989, pp110-115.
- [5] Li Gong, "On security in capability-based systems", Operating System Review (ACM) vol. 23, no. 2, 1989, pp56-60.
- [6] C. S. Laih, L. Ham, J. Y. Lee, "On the design of a single-key-lock mechanism based on Newton's interpolating polynomial", IEEE Trans. on software Engineering vol. 15, no. 9, 1989, pp1135-1137.
- [7] R. S. Sandhu, "Nested categories for access control", Computer and Security vol. 7, no. 6, 1988, pp599-605.
- [8] S. T. Vinter, "EXTENDED DISCRETIONARY ACCESS CONTROL", Proceedings of Security and Privacy, 1988, p39-49.
- [9] G. Montai, S. Sirovich, "ACCESS CONTROL MODELS AND OFFICE STRUCTURE", Proceedings of the Second IFIP International Conference on Computer Security, IFIP/Sec'84, pp473-485.
- [10] K. Swaminathan, "Negotiated Access Control", Proceedings of Security and Privacy, 1985, pp190-196.



- [11] A. Silberschatz, J. Peterson and J. Galvin, "Operating System Concept", 3rd Edition, Addison Wesley, 1991.
- [12] David A. Taylor, "Object-Oriented Technology: A Manager's Guide", Addison Wesley, 1992.
- [13] R.G.G. Cattell, "Object Data Management", Addison Wesley, 1991.
- [14] Charles P. Pfleeger, Security in Computing, Prentice-Hall International Editions, 1989

# Supporting Multiple Access Control Policies in Database Systems

Elisa Bertino

Dipartimento di Scienze dell'Informazione  
Università degli Studi di Milano  
20135 Milano (Italy)

Sushil Jajodia\*

The MITRE Corporation  
1820 Dolley Madison Boulevard  
McLean, VA 22102-3481 (USA)

Pierangela Samarati

Dipartimento di Scienze dell'Informazione  
Università degli Studi di Milano  
20135 Milano (Italy)

## Abstract

*Although there are several choices of policies for protection of information, access control models have been developed for a fixed set pre-defined access control policies that are then built into the corresponding access control mechanisms. This becomes a problem, however, if the access control requirements of an application are different from the policies built into a mechanism. In most cases, the only solution is to enforce the requirements as part of the application code, but this makes verification, modification, and adequate enforcement of these policies impossible. In this paper, we propose a flexible authorization mechanism that can support different security policies. The mechanism enforces a general authorization model onto which multiple access control policies can be mapped. The model permits negative and positive authorizations, authorizations that must be strongly obeyed and authorizations that allow for exceptions, and enforces ownership together with delegation of administrative privileges.*

## 1 Introduction

Recent years have witnessed considerable work on access control models and related mechanisms for databases [8]. Most of this work deals with relational

databases systems, although recently advances have been reported in the areas of object-oriented database systems [3] and deductive databases [6]. Despite this large research effort, current access control models and mechanisms are not flexible enough to meet the access control requirements of modern application environments [10]. This lack of flexibility exists because each model has been developed for a number of pre-defined access control policies; these policies are built into the corresponding access control mechanism and, therefore, cannot be changed. The majority of the models are based on a *closed world* policy, which permits specification of only positive authorizations and allows only accesses explicitly authorized. In contrast, very few models are based on the *open world* policy which permits the specifications of only negative authorizations, and all accesses not explicitly denied are allowed. Recent authorization models enforcing a closed policy also allow users to specify negative authorizations and resolve conflicts according to some predefined rules [5].

A problem arises when the access control requirements of an application differ from the policies built into the mechanism at hand. Enforcing the application policy becomes very difficult and, in most cases, the only solution is to implement the policy as part of the application code. This solution, however, makes the tasks of verification, modification, and adequate enforcement of the the policy difficult.

A promising solution to this problem is represented by the *multipolicy* access control systems. The development of a multipolicy system is a challenging area that is still at an early stage. It poses several challenges, including (1) what is the proper spectrum of policies

\*The work of S. Jajodia was partially funded by MITRE Sponsored Research, project 91850. He is also with the Center for Secure Information Systems and Department of Information and Software Systems Engineering, George Mason University, Fairfax, VA 22030-4444

with respect to which the users must be able to tailor the system, (2) how to decouple the policy from the enforcement mechanism, (3) what is the efficiency of the mechanism, and (4) whether conflicting policies can be defined for the same data objects and, if so, how to resolve such conflicts.

In this paper, we take a first step toward the investigation of these issues. We present a multipolicy system that can support a number of discretionary access control policies for relational database systems, including the traditional closed and open policies, closed policy with negative authorizations and the application of the denials-take-precedence principle (to resolve conflicts among authorizations), and the closed policy with negative authorizations and the enforcement of the most specific authorization takes precedence principle.

The architecture of the system has two components: an access control mechanism and a mediator. The authorization model enforced by the mechanism is actually based on the closed policy with negative authorizations where conflicting authorizations are resolved by application of the most specific authorization takes precedence principle. The mediator, which is an interface between users and the access control mechanism, supports other policies by using only some of the features of the authorization model. An important feature of the system architecture is that it has been designed to separate the policy specification component from the authorization and enforcement components of the system. Here, we focus on the authorization model and show how its flexibility can be exploited to support a number of different policies within the same database.

The remainder of this paper is organized as follows. We begin in Section 2 by giving the architecture of the system. Section 3 illustrates the basic elements of the authorization model. Section 4 discusses the access control mechanism, which shows how the system determines whether to grant or reject access requests. Section 5 discusses administration of authorizations. Sections 6 and 7 discuss the consistency of the authorization state and the resolution of conflicts among authorizations, respectively. Section 8 illustrates how the mediator and the model proposed in this paper are used to support different policies. Section 9 discusses related work. Finally, Section 10 presents the conclusions.

## 2 Architecture of the multipolicy access control system

The goal of our system is to provide a flexible framework that can support different protection policies. Upon creation of each table, its owner can specify the type of protection policy he or she wishes to apply to

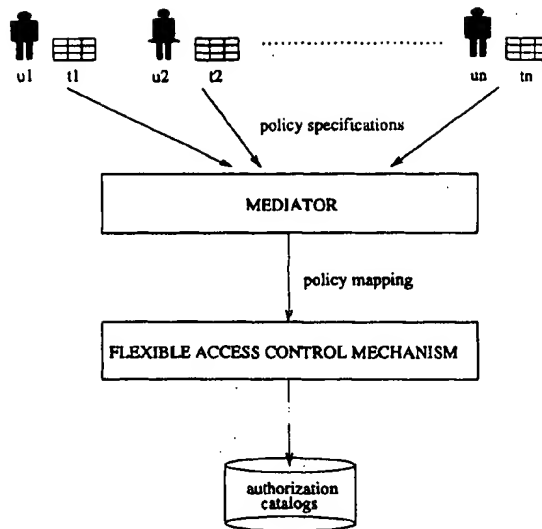


Figure 1. Multipolicy access control system architecture

that specific table. What is interesting is that it is possible to protect different tables in the same database according to different policies.

The architecture of our system is illustrated in Figure 1. The two major components of the architecture are i) the access control mechanism implementing the authorization model and ii) a mediator [24], which is an interface between users and the access control mechanism. The model enforced by the access control mechanism is a general model able to support different protection policies (see Section 8). It is a task of the mediator to map the different protection policies and protection requirements onto a set of authorizations in the general model implemented by the mechanism. The mediator also filters every grant request to ensure that authorizations granted obey the specific policy for the object. For example, the mediator must ensure that no positive authorization can be specified on an object on which an open policy is to be applied.

The authorization model that is implemented by the access control mechanism is described in the next section. It is built on the concept of strong and weak authorizations first proposed in the Orion authorization model [19]. However, our model differs considerably from Orion not only in terms of the semantics associated with negative authorizations, but also in the way conflicts between authorizations are resolved. Moreover, unlike Orion, our model supports ownership to

gether with decentralized administration of authorizations. At the same time, administrative privileges can also be restricted in our model so that owners retain control over their tables.

### 3 The authorization model

In this section, we introduce our authorization model. We start by giving basic notations and definitions and then discuss how negative and positive authorizations coexist in our model.

#### 3.1 Notations and definitions

Subjects of authorizations can be either users or groups. Members of groups can be users as well as other groups. Hence, membership of a subject  $s$  in a group  $G_k$  can be either *direct* or *indirect*. The membership is direct, written  $s \in_1 G_k$ , if the subject is defined as a member of  $G_k$ . The membership is indirect, written  $s \in_n G_k, n > 1$ , if there exists a sequence of subjects  $\langle s_1, \dots, s_{n+1} \rangle$ , such that  $s_1 = s, s_{n+1} = G_k$ , and  $s_i \in_1 s_{i+1}, 1 \leq i \leq n$ . Sequence  $\langle s_1, \dots, s_{n+1} \rangle$  is called a *membership path* of  $s$  to  $G_k$ , written  $mp(s, G_k)$ . We use  $\in_0$  to indicate equality, i.e., for each subject  $s$ ,  $s \in_0 s$ .<sup>1</sup> In the following discussion, we use the term *membership* to refer to either direct or indirect membership. We will explicitly use the terms direct membership or indirect membership when a distinction is needed. A subject can belong to a group in many ways, either as a direct or as an indirect member. We use  $MP(s, G_k)$  to denote the set of all membership paths from user  $s$  to group  $G_k$ .

The membership relationship between subjects can be represented by a graph, called the *group membership graph*, in which nodes represent subjects and an edge from node  $s_i$  to node  $s_j$  indicates that  $s_i$  is a direct member of  $s_j$  (i.e.,  $s_i \in_1 s_j$ ). We require that the group membership graph be a directed acyclic graph.

Authorizations specify the privileges that subjects are authorized or denied on objects. In our model, a grantor has the option of specifying whether the authorization (positive or negative) he is granting can be overridden or not. To support this functionality, we introduce, as in [19], the notion of *strong* and *weak* authorizations. The basic idea behind this approach is that strong authorizations cannot be overridden, while weak authorizations can be overridden, according to specified rules,<sup>2</sup> by other strong or weak authorizations.

<sup>1</sup>We continue to use the symbol  $\in$  to denote the membership of an element in a set.

<sup>2</sup>These rules are given in the next subsection.

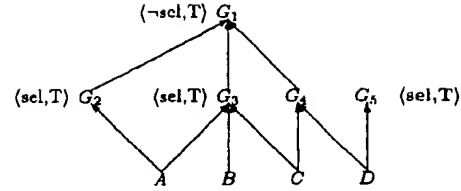


Figure 2. An authorization state

Let  $U$  denote the users in the system,  $G$  the set of groups,  $P$  the set of privileges (i.e., select, insert, delete, and update),  $T$  the set of tables, and  $S = U \cup G$  the set of all subjects (either users or groups) in the system. Authorizations are defined as follows.

**Definition 1 (Authorization)** An authorization is a 6-tuple of the form  $\langle s, p, pt, t, g, at \rangle$ , with  $s \in S$ ,  $p \in P$ ,  $pt \in \{+, -\}$ ,  $t \in T$ ,  $g \in U$ , and  $at \in \{\text{weak}, \text{strong}\}$ .

Authorization  $\langle s, p, pt, t, g, at \rangle$  states that  $s$  has been granted (denied if  $pt = -$ ) privilege  $p$  on table  $t$  by user  $g$ . If  $at = \text{weak}$  the authorization admits exceptions. For example, tuple  $\langle G_2, \text{select}, -, T, A, \text{weak} \rangle$  indicates that members of group  $G_2$  cannot select tuples from table  $T$ , and this authorization, which admits exceptions, was granted by user  $A$ .

Given an authorization  $a$ , we use the notation  $s(a)$ ,  $p(a)$ ,  $pt(a)$ ,  $t(a)$ ,  $g(a)$ ,  $at(a)$  to denote the subject, the privilege, the privilege type, the table, the grantor, and the authorization type in  $a$ , respectively. For example,  $s(a)$  is the subject in authorization  $a$ .

The set of authorizations that are present at a given time represents the *authorization state*, denoted by  $AS$ . In the following discussion, we graphically represent the authorization state by associating authorizations with subjects in the group membership graph. We indicate that subject  $s_i$  owns a positive authorization for privilege  $p$  on table  $t$  by associating a pair  $(p, t)$  with node  $s_i$  in the group membership graph. Analogously, pair  $(\neg p, t)$  associated with node  $s_i$  indicates that  $s_i$  owns a negative authorization for  $p$  on  $t$ . To distinguish between strong and weak authorizations, we write strong authorizations in bold type. An example of authorization state is illustrated in Figure 2.

A user is allowed to exercise, beside his own authorizations, all the authorizations of any of the groups to which he belongs. We do not restrict the user to the use of the authorizations of only one group at a time. A user has all the privileges that are in the union of all his personal authorizations and the authorizations of

all the groups to which he belongs. Note that this approach follows the approach for group management of most existing database management systems (DBMSs) like, for example, System R [25, 13] and Orion [19]. This approach accords with the concept of user group as opposed to the concept of role (where a user may be constrained to operate with the privileges of a single role at a time) [1, 20].

Complications arise when a user is both authorized and denied, either directly or indirectly (i.e., through a group), for a privilege on a table. In the following discussion, we give our approach for dealing with the simultaneous presence of positive and negative authorizations.

### 3.2 Overriding authorizations

When dealing with authorizations admitting exceptions, an important issue is to devise proper overriding rules. Such rules establish those circumstances under which exceptions can be issued against a weak authorization. In our model, strong authorizations always override weak authorizations. The overriding rule between weak authorizations is based on the concept of more specific authorization. Authorizations given to a member of a group are always more specific than the authorizations given to the group (with respect to the member). The overriding rule between authorizations is formalized as follows:

**Definition 2 (Overriding authorization)** An authorization  $a_i$  overrides a weak authorization  $a_j$ , with  $p(a_i) = p(a_j), t(a_i) = t(a_j), pt(a_i) \neq pt(a_j)$ , with respect to subject  $s$  (written  $a_i \triangleright_s a_j$ ), with  $s \in_n s(a_i), s \in_m s(a_j)$ , for some  $n, m \geq 0$ , iff any of the following conditions is satisfied:

- $at(a_i) = \text{strong}$
- $at(a_i) = \text{weak}, s = s(a_i), s \neq s(a_j)$
- $at(a_i) = \text{weak}, s(a_i) \in_l s(a_j), l \geq 1$ , and  $\forall mp \in MP(s, s(a_j))$ : either  $s(a_i) \in mp$ , or  $\exists s' \in S, a' \in AS, \exists k \geq 0$  such that  $s' \neq s, s' \neq s(a_j), s' \in_k s(a_i), a' \triangleright_{s'} a_j$ .

Definition 2 states that:

- a strong authorization  $a_i$  overrides a weak authorization  $a_j$ , with same privilege and table, but different privilege type, with respect to any subject which belongs to both  $s(a_i)$  and  $s(a_j)$ ;
- a weak authorization  $a_i$  overrides a weak authorization  $a_j$ , with same privilege and table, but different privilege type, with respect to subject  $s(a_i)$ , if  $s(a_i)$  belongs to  $s(a_j)$ ;

- a weak authorization  $a_i$  overrides a weak authorization  $a_j$  with same privilege and table, but different privilege type, and with  $s(a_i)$  that belongs to  $s(a_j)$ , with respect to subject  $s$ , which belongs to both  $s(a_i)$  and  $s(a_j)$  if and only if every membership path from  $s$  to  $s(a_j)$  either contains  $s(a_i)$  or contains a subject  $s'$ , not belonging to  $s(a_i)$ , with respect to which authorization  $a_j$  is overridden.<sup>3</sup>

Notice that the definition of overriding authorizations is given in terms of a subject since the outcome of the override operation may be different depending on the subject. Even if authorization  $a_i$  overrides authorization  $a_j$  with respect to a group  $G$ , it does not follow that  $a_i$  will override  $a_j$  with respect to every subject  $s$  which is a member of  $G$ .

**Example 1** Consider the authorization state illustrated in Figure 2. The negative weak authorization  $(\neg \text{sel}, T)$  specified for  $G_1$  is overridden by (i) strong positive authorization  $(\text{sel}, T)$  specified for  $G_5$  with respect to user  $D$ , (ii) positive weak authorization  $(\text{sel}, T)$  specified for  $G_2$  with respect to group  $G_2$  as well as user  $A$ , and (iii) the positive weak authorization  $(\text{sel}, T)$  specified for  $G_3$  with respect to group  $G_3$  as well as users  $A$  and  $B$ . Notice that it is not overridden by the latter authorization with respect to user  $C$ . This is because  $C$  belongs to  $G_1$  also due to its membership of  $G_4$ , for which no authorization is specified.  $\square$

### 3.3 Conflicts among authorizations

Two contrasting authorizations such that neither one of them overrides the other are said to be in conflict. This is formalized as follows:

**Definition 3 (Conflicting authorizations)** Two authorizations  $a_i$  and  $a_j$  conflict with respect to subject  $s$  (written  $a_i \diamond_s a_j$ ), with  $s \in_n s(a_i), s \in_m s(a_j)$ , for some  $n, m \geq 0$  iff  $p(a_i) = p(a_j), t(a_i) = t(a_j), pt(a_i) \neq pt(a_j)$  and neither  $a_i \triangleright_s a_j$  nor  $a_j \triangleright_s a_i$ .

For instance, with reference to the authorization state illustrated in Figure 2, the authorizations of  $G_1$  and  $G_3$  conflict over user  $C$ . Note that the concept of conflicting authorization is also defined with respect to a subject since it is being defined in terms of the overriding relationship.

According to Definition 2, since a strong authorization always overrides a weak authorization (with respect to a given subject), no conflicts can exist between a strong authorization and a weak authorization.

<sup>3</sup> Authorization  $a_i$  and the authorization specified for  $s'$  "collaborate" in overriding  $a_j$ .

Therefore, conflicts may exist only between two strong authorizations or between two weak authorizations.

If two strong authorizations exist which conflict with respect to a given subject, we say that the authorization state is *inconsistent*. It is considered *consistent* otherwise. This is formalized as follows:

**Definition 4 (Consistency of authorization state)**  
*The authorization state is consistent iff strong authorizations do not conflict (with respect to any subject).*

In our model, the simultaneous presence of conflicting strong authorizations is not permitted. This is because we wish to ensure that the semantics of strong authorizations will always be respected. Then, we require the following invariant to hold.

**Invariant 1** *The authorization state is consistent.*

It is the task of the access control system to ensure the consistency of the authorization state. Section 6 discusses how the system operates to ensure consistency of the authorization state.

In our model, we accept the simultaneous presence of two conflicting weak authorizations. This choice is justified by the following facts: i) requiring complete absence of conflicts may make the authorization specification task really difficult, and ii) conflicts between weak authorizations do not contrast with the requirements of the users specifying the authorizations.

As we will see in the following section, whenever two weak authorizations conflict over a given subject with respect to a given access, we consider both authorizations to be invalid for the subject. This means that the positive authorization cannot be applied and therefore the access will be denied to the subject. This policy is safe since the access is denied in case of a conflict. In Section 7, we will present an approach for solving conflicts between weak authorizations.

## 4 Access control mechanism

In this section, we illustrate how, given an access request, the access control mechanism determines whether to grant or reject the request.

An access request is characterized as a triple  $\langle u, p, t \rangle$  stating that user  $u$  is requesting to exercise privilege  $p$  on table  $t$ . The access request is authorized iff there exists a positive authorization for the access that is neither overridden nor conflicting with respect to  $u$ . This is formalized by the following definition.

**Definition 5 (Authorized access request)** *An access request  $\langle u, p, t \rangle$ ,  $u \in U$ ,  $p \in P$ ,  $t \in T$ , is authorized iff  $\exists a \in AS$ ,  $n \geq 0$ ,  $\nexists a_i \in AS$  such that  $u \in_n s(a)$ ,  $p = p(a)$ ,  $t = t(a)$ ,  $pt(a) = "+"$ , and  $(a_i \circ_u a \vee a_i \triangleright_u a)$ .*

Positive authorizations which are overridden or conflicting with other authorizations with respect to the user are said to be *invalid* for the user.

Suppose a user requests to access a table. If there is a positive strong authorization for the required access, by Invariant 1, a strong negative authorization cannot exist for the same access. Moreover, any weak negative authorization for the required access will be overridden by the strong authorization. Hence the access must be authorized. Analogously, if there is a strong negative authorization for the access, the access must be denied. If there are no strong authorizations for the required access, the weak authorizations must be evaluated. The access must be authorized if a positive weak authorization exists which is neither overridden nor conflicting over  $u$ ; otherwise, it must be denied.

We view the access control as a function that, given an access request, returns **true** if the request must be authorized and **false** otherwise. Function `access_control()` can be expressed as follows.

$$\text{access\_control}(u, p, t) = \begin{cases} \text{str\_auth}(u, p, t) & \text{if str\_auth}(u, p, t) \neq \text{undecided} \\ \text{weak\_auth}(u, p, t) & \text{otherwise} \end{cases}$$

where:

$$\text{str\_auth}(u, p, t) = \begin{cases} \text{true} & \text{if } \exists \text{ a strong positive authorization for the access} \\ \text{false} & \text{if } \exists \text{ a strong negative authorization for the access} \\ \text{undecided} & \text{otherwise} \end{cases}$$

$$\text{weak\_auth}(u, p, t) = \begin{cases} \text{true} & \text{if } \exists \text{ a positive weak authorization for the access that is neither overridden by nor conflicting with other authorizations over } u \\ \text{false} & \text{otherwise} \end{cases}$$

The algorithms implementing functions `str_auth()` and `weak_auth()` are given in Figures 3 and 4, respectively.

The algorithm in Figure 3, implementing function `str_auth()`, works as follows. First, the authorizations of the user requiring the access are examined (step 1). If any strong authorization is found for the user with respect to the access, then the search is terminated. Hence, if the authorization is a positive authorization, **true** is returned; otherwise, **false** is returned. If no authorization is specified for the user, then the authorizations of the groups to which the user belongs are

---

**str\_auth(u,p,t)**

1. Look for an authorization  $a \in AS$  such that  $s(a) = u$ ,  $p(a) = p$ ,  $t(a) = t$ ,  $at(a) = \text{strong}$ .  
If such an  $a$  is found then go to step 3.
  2.  $GS := \{G_i \mid u \in_i G_i \text{ for some } i \geq 1\}$ .  
If  $GS = \emptyset$  then return *undecided* and exit.  
Look for an authorization  $a \in AS$  such that  $s(a) \in GS$ ,  $p(a) = p$ ,  $t(a) = t$ ,  $at(a) = \text{strong}$ .  
If such an authorization is found then go to step 3  
else return *undecided* and exit.
  3. If  $p(a) = "+"$  then return *true* else return *false*.
- 

**Figure 3. Algorithm for function str\_auth()**

evaluated (step 2). If a strong authorization for the access is found for any of the groups to which the user belongs, the search is terminated. Hence, if the authorization found is a positive authorization, *true* is returned; otherwise, *false* is returned. By contrast, if no strong authorization is found, *undecided* is returned.

Since weak authorizations may conflict or be overridden, evaluation of the function *weak\_auth()* may require examination of all authorizations applicable to the user through different membership paths connecting him to all the groups to which he belongs. However, it is not necessary to traverse all membership paths fully since more specific authorizations override less specific authorizations. Thus, first the authorizations of the user are considered. Then, the authorizations of the groups to which the user belongs are considered traversing the membership paths in increasing distance. Whenever an authorization is found for a group, it is not necessary to proceed further on that membership path since either this authorization overrides those of the groups above it on that membership path or these groups are reachable through some other membership paths.

The algorithm implementing function *weak\_auth()*, illustrated in Figure 4, works as follows. If the user owns a negative authorization for the access, then *false* is returned (step 1) and the algorithm terminates. Otherwise, if the user owns a positive authorization, then *true* is returned (step 2) and the algorithm terminates.<sup>4</sup> If no authorization for the access is specified for the user, the authorizations of the groups to which the user belongs are examined. The groups to which the user directly belongs are considered in step 3. If any of these groups owns a negative authorization for the access, then *false* is returned (step 4) and the algorithm terminates. If not, the sys-

<sup>4</sup>Note that, since weak authorizations may conflict it is important to look first for a negative authorization and then, only if no negative authorization is found, look for a positive authorization.

---

**weak\_auth(u,p,t)**

1. Look for an authorization  $a \in AS$  such that  $s(a) = u$ ,  $p(a) = p$ ,  $t(a) = t$ ,  $pt(a) = "-"$ ,  $at(a) = \text{weak}$ .  
If such an authorization exists then return *false* and exit.
  2. Look for an authorization  $a \in AS$  such that  $s(a) = u$ ,  $p(a) = p$ ,  $t(a) = t$ ,  $pt(a) = "+"$ ,  $at(a) = \text{weak}$ .  
If such an authorization exists then return *true* and exit.
  3.  $Groups := \{G_i \mid u \in_i G_i\}$
  4. Look for an authorization  $a \in AS$  such that  $s(a) \in Groups$ ,  $p(a) = p$ ,  $t(a) = t$ ,  $pt(a) = "-"$ ,  $at(a) = \text{weak}$ .  
If such an authorization exists then return *false* and exit.
  5.  $A\_Groups := \{G_k \in Groups \mid \exists a \in AS, s(a) = G_k, p(a) = p, t(a) = t, pt(a) = "+", at(a) = \text{weak}\}$
  6.  $N\_Groups := Groups - A\_Groups$
  7. If  $N\_Groups \neq \emptyset$  then  $Groups := \{G_j \mid \exists G \in N\_Groups, G \in_i G_j\}$  else go to step 9
  8. If  $Groups \neq \emptyset$  go to step 4
  9. If  $A\_Groups \neq \emptyset$  then return *true* else return *false*.
- 

**Figure 4. Algorithm for function weak\_auth()**

tem checks whether one of these groups owns a positive authorization for the access (step 5). If a group owns a positive authorization for the access, then it is no longer necessary to look at the authorizations of other groups along the same membership path. Let  $N\_Groups$  denote the groups for which no authorization has been found. Next, the groups to which any group in  $N\_groups$  directly belongs are considered (step 7). If such groups do exist, the process of looking for a weak authorization for the access is repeated (step 8); otherwise, the algorithm terminates. A *true* is returned if any positive authorization was found during the execution; otherwise, *false* is returned.

**Example 2** Consider the authorization state illustrated in Figure 5, and consider the requests by user  $B$  to exercise the select privilege on table  $T$ . Function *str\_auth(B,sel,T)* returns *undecided* and therefore function *weak\_auth(B,sel,T)* is called. This function returns *false* due to the negative authorization of  $G_4$  and hence the access is denied.  $\square$

## 5 Administration of authorizations

The user creating an object is considered the owner of the object. As owner, the user can exercise any privilege on the table and can grant other subjects any authorization on the table. Moreover, the owner can also delegate to other subjects the administration of

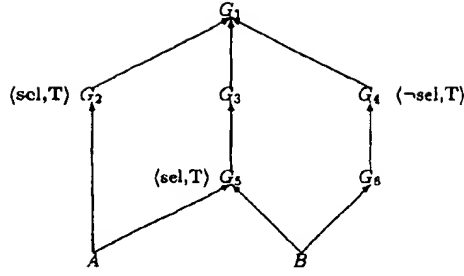


Figure 5. An authorization state

some privileges on the table. Delegation of administration is enforced through two administrative privileges: *adm-access* and *administer*. The *adm-access* privilege for a privilege on a table allows subjects to grant authorizations for the privilege on the table. The *administer* privilege for a privilege on a table allows subjects to grant access authorizations and administrative privileges for the privilege on the table. We do not apply the strong and weak classification on administrative authorizations. Allowing administrative authorization to be overridden would raise the problem of dealing with authorizations granted by a user whose administrative authorizations become overridden: if the authorizations of the user are not valid, neither should the authorizations he granted. This propagation of the "overriding" effect would make the authorization administration task unnecessarily complicated. Therefore, we take the approach that administrative authorizations cannot be overridden. To ensure this, every time a subject is granted an administrative authorization for a privilege on a table, a strong authorization for the privilege on the table is also granted by the system to the subject. Upon deletion of the administrative authorizations, this strong authorization will also be deleted.

Although we do not distinguish between administrative authorizations themselves as strong and weak, we allow users to grant administrative privileges with the restriction that only weak authorizations can be granted.

Administrative authorizations are defined as follows.

**Definition 6 (Administrative authorization)**

An administrative authorization  $a$  is a 6-tuple  $\langle s, p, ap, gat, t, g \rangle$  with  $s \in S$ ,  $p \in P$ ,  $ap \in \{\text{adm-access}, \text{administer}\}$ ,  $gat \in \{\text{strong}, \text{weak}\}$ ,  $t \in T$ ,  $g \in U$ .

Administrative authorization  $\langle s, p, ap, gat, t, g \rangle$  states

that subject  $s$  has administrative privilege  $ap$  on access mode  $p$  on table  $t$  and that this authorization was granted by user  $g$ . If  $gat = \text{weak}$ ,  $s$  can grant only weak authorizations.

For instance, authorization  $\langle A, \text{select}, \text{adm-access}, \text{weak}, T_1, B \rangle$  states that  $A$  can grant, and revoke, other subjects' weak authorizations for the select privilege on table  $T_1$  and that this authorization was granted by  $B$ .

Administrative authorizations are summarized in Table 1. Note that, since granting an administrative authorization to a subject implies granting a strong positive authorization to the subject, authorizations for the *administer* privilege must be necessarily strong.

Note also that subjects holding an administrative privilege can grant positive as well as negative authorizations. This choice is justified by the fact that, since negative/positive authorizations can be used to specify exceptions to other positive/negative authorizations, users allowed to grant permissions should also be able to specify negations and vice versa. However, the model can be easily extended to the consideration of different privileges for the administrations of permissions and denials by adding the privilege type (+ or -) in the administrative authorization.

Subjects holding administrative privileges can also revoke authorizations. A user can revoke only authorizations he has granted. Moreover, the authorizations granted by a user can exist only as far as the user has the corresponding administrative privilege. As a consequence, when a user is revoked the administrative authorization for a privilege on a table, a recursive revocation may take place to delete the authorizations granted by the user or derived for the users on views. Revocation algorithms enforcing recursive deletion of authorizations proposed in other models [4, 11, 13, 15, 23] can be easily adapted to our model.

## 6 Ensuring consistency of the authorization state

The collection of valid authorizations may change upon execution of administrative operations by the users. These operations include changes to the authorization state, the group membership graph, or the tables. Operations affecting the authorization state are grant and revoke of privileges on tables, operations affecting the group membership graph are the addition and removal of members from groups, and operations affecting the tables are the creation and deletion of tables. Although some of these operations do not have any effect on the authorizations themselves, they may affect the consistency of the authorization state.



| Administrative authorization                                   | Semantics   |
|--|---|
| $\langle s, p, \text{adm-access}, \text{strong}, t, g \rangle$ | $s$ can grant and revoke weak/strong positive/negative access authorizations for $p$ on $t$   |
| $\langle s, p, \text{adm-access}, \text{weak}, t, g \rangle$   | $s$ can grant and revoke weak positive/negative access authorizations for $p$ on $t$  |
| $\langle s, p, \text{administer}, \text{strong}, t, g \rangle$ | $s$ can grant and revoke weak/strong positive/negative access authorizations for $p$ on $t$<br>$s$ can grant and revoke weak/strong authorizations for the administration of $p$ on $t$ |
| $\langle s, p, \text{administer}, \text{weak}, t, g \rangle$   | N/A   |

Table 1. Administrative authorizations and their semantics

Hence, every time an administrative operation is requested, beside the existence of the authorization necessary for the execution of the request, the consistency of the resulting authorization state must be checked. If the operation would result in an inconsistent authorization state, its execution must be refused by the system.

Note that operations that decrease authorizations applicable to the subjects (i.e., revocation of authorizations, removal of members from groups, or deletion of tables) do not affect consistency. By contrast, operations increasing the authorizations applicable to the subjects may result in an inconsistent authorization state and must therefore be controlled.

Assignment of authorizations to the owner upon creation of new tables does not obviously affect consistency (no authorization may exist on the table before it is created). Let us therefore consider the operations of granting a new strong authorization and adding a new member to a group.

### 6.1 Granting of new strong authorizations

Suppose a user wishes to insert a new strong authorization  $a$ . Inconsistencies may arise if subject  $s(a)$ , any member of  $s(a)$ , or any group to which  $s(a)$  or any of its members belongs, already owns a strong authorization for privilege  $p(a)$  on table  $t(a)$  with privilege type different from  $pl(a)$ .

For instance, consider the authorization state shown in Figure 6. Granting a new strong authorization  $\langle \neg \text{sel}, T_3 \rangle$  to  $G_3$  would create a conflict with the strong authorization  $\langle \text{sel}, T_3 \rangle$  specified for  $G_3$ . Granting a new strong authorization  $\langle \text{sel}, T_2 \rangle$  to  $G_3$  would create a conflict with respect to subject  $G_3$  and all its members with strong authorization  $\langle \neg \text{sel}, T_2 \rangle$  specified for  $G_2$ . Granting a new strong authorization  $\langle \text{sel}, T_4 \rangle$  to  $G_3$  would create a conflict with respect to user  $A$ , with the strong authorizations  $\langle \neg \text{sel}, T_4 \rangle$  specified for  $A$ .

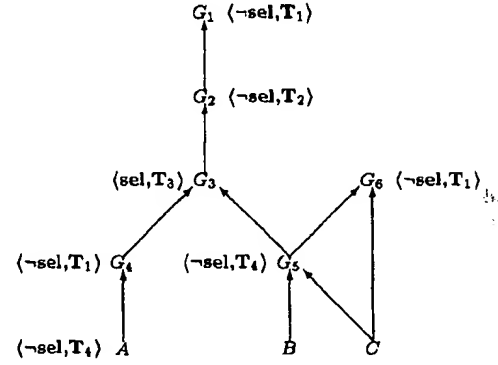


Figure 6. An authorization state

Moreover, it would create a conflict, with respect to subject  $G_5$  and all its members with strong authorization  $\langle \neg \text{sel}, T_4 \rangle$  specified for  $G_5$ .

Figure 7 illustrates an algorithm for determining whether the addition of a new authorization would result in an inconsistent authorization state. The algorithm returns ok if the resulting authorization state is consistent; it returns the conflicts which would arise otherwise. If there is a conflict between the new authorization and another authorization with respect to subject  $s$  only because of the membership of  $s$  to another subject  $s'$  with respect to which the two authorizations conflict, then only the conflict over subject  $s'$  is returned. Conflicts which are implied by other conflicts are not returned for sake of simplicity (it is sufficient to communicate to the user the conflict which implies the other ones). For example, with reference to Figure 6, if authorization  $\langle \text{sel}, T_4 \rangle$  is being granted to  $G_3$ , the conflict with the negative authorization specified for subject  $G_5$  with respect subject  $G_5$  is returned. Although these authorizations also conflict with respect

to all members of  $G_5$  (viz.,  $B$  and  $C$ ), these conflicts are not returned.

---

**Check\_autho(a)**

1.  $Anc := \{s \in S \mid \exists n \geq 1, s(a) \in_n s\}$
  2.  $Desc := \{s \in S \mid \exists n \geq 0, s \in_n s(a)\}$
  3.  $Anc\_Desc := \{s \in G - (Anc \cup Desc) \mid \exists n \geq 1, s_i \in Desc : s_i \in_n s\}$
  4.  $Confl\_Anc := \{a_i \in AS \mid s(a_i) \in Anc, p(a_i) = p(a), t(a_i) = t(a), pt(a_i) \neq pt(a), at(a_i) = strong\}$
  5.  $Confl\_Desc := \{a_i \in AS \mid s(a_i) \in Desc, p(a_i) = p(a), t(a_i) = t(a), pt(a_i) \neq pt(a), at(a_i) = strong\}$
  6.  $Confl\_AD := \{a_i \in AS \mid s(a_i) \in Anc\_Desc, p(a_i) = p(a), t(a_i) = t(a), pt(a_i) \neq pt(a), at(a_i) = strong\}$
  7. If  $Confl\_Anc \cup Confl\_Desc \cup Confl\_AD = \emptyset$  then return ok and exit
  8. For each  $a_i \in Confl\_Anc$  do return "conflict:  $a \circ_{s(a)} a_i$ " endfor
  9. For each  $a_i \in Confl\_Desc$  do return "conflict:  $a \circ_{s(a)} a_i$ " endfor
  10. For each  $a_i \in Confl\_AD$  do  
 $Confl(a_i) := \{s \in S \mid \exists n, m \geq 0, s \in_n s(a), s \in_m s(a_i)\}$   
 $Disj\_Confl(a_i) := \{s \in Confl(a_i) \mid \exists mp \in (MP(s, s(a))) \cup MP(s, s(a_i)), \exists s', s' \neq s, s' \in Confl(a_i), s' \in mp\}$   
For each  $s \in Disj\_Confl(a_i)$  do return "conflict:  $a \circ_s a_i$ " endfor  
endfor.
- 

**Figure 7. Evaluation of the consistency of the authorization state resulting from addition of authorization  $a$**

The algorithm in Figure 7 works as follows. Suppose the addition of a new strong authorization  $a$  is required. In step 1, set  $Anc$  consisting of all subjects (groups) to which subject  $s(a)$  belongs is determined. In step 2, set  $Desc$  consisting of all subjects which belong to  $s(a)$ , including  $s(a)$  itself, is determined. In step 3, set  $Anc\_Desc$  consisting of all groups (not already in  $Anc$  or in  $Desc$ ) to which any of the members of  $s(a)$ , or  $s(a)$  itself, belongs is determined. In step 4, set  $Confl\_Anc$  of strong authorizations specified for any group in  $Anc$  with same privilege and table as authorization  $a$  being granted but with different privilege type is determined. Analogously, in step 5 and 6, sets  $Confl\_Desc$  and  $Confl\_AD$  of strong authorizations specified for any subject in  $Desc$  and in  $Anc\_Desc$ , respectively, with same privilege and table as authorization  $a$  being granted but with different privilege type are determined. If all three sets,  $Confl\_Anc$ ,  $Confl\_Desc$ , and  $Confl\_AD$ , are empty, a conflict is not found and ok is returned (step 7). Otherwise, conflicts which would have been introduced if authorization  $a$  were inserted are returned as follows.

In step 8, the conflicts between authorization  $a$  being inserted and every authorization  $a_i \in Confl\_Anc$  with respect to subject  $s(a)$  for which the authorization is being inserted is returned. In step 9, the conflict between authorization  $a$  being inserted and every authorization  $a_i \in Confl\_Desc$  with respect to subject  $s(a_i)$  of the considered conflicting authorization is returned. Finally, in step 10 the conflicting authorizations specified for other groups to which the members of  $s(a)$  belongs are considered. To avoid returning a conflict with respect to a subject which belongs to  $s(a)$  and  $s(a_i)$  only because of its membership of another subject for which the conflict is also returned, the following process is used. For each authorization  $a_i \in Confl\_AD$ , set  $Confl(a_i)$  of all the subjects with respect to which  $a_i$  conflicts with  $a$ , i.e., all subjects belonging to both  $s(a)$  and  $s(a_i)$ , is determined. Then set  $Disj\_Confl(a_i)$  of all subjects in  $Confl(a_i)$  for which there exists a membership path to  $s(a)$  or to  $s(a_i)$  which does not contain any other subject in  $Confl(a_i)$  is determined. Finally, for each subject  $s \in Disj\_Confl(a_i)$ , the conflict of authorization  $a$  being inserted with authorization  $a_i$ , with respect to subject  $s$ , is returned.

To illustrate how the algorithm works, let us consider the following example. In the following  $a_{s,t}$  denotes an authorization (either positive or negative) of subject  $s$  on table  $t$

**Example 3** Consider the authorization state illustrated in Figure 6 and suppose that a new strong positive authorization for the select privilege on table  $T_1$  is being granted to group  $G_3$ . The algorithm works as follows:

1.  $Anc := \{G_1, G_2\}$
2.  $Desc := \{G_3, G_4, G_5, A, B, C\}$
3.  $Anc\_Desc := \{G_6\}$
4.  $Confl\_Anc := \{a_{G_1, T_1}\}$
5.  $Confl\_Desc := \{a_{G_4, T_1}\}$
6.  $Confl\_AD := \{a_{G_6, T_1}\}$
7.  $\{a_{G_1, T_1}, a_{G_4, T_1}, a_{G_6, T_1}\} \neq \emptyset$  continue
8. return "conflict:  $a_{G_3, T_1} \circ_{G_3} a_{G_1, T_1}$ "
9. return "conflict:  $a_{G_3, T_1} \circ_{G_4} a_{G_4, T_1}$ "
10.  $Confl(a_{G_6, T_1}) := \{G_5, B, C\}$   
 $Disj\_Confl(a_{G_6, T_1}) := \{G_5, C\}$   
return "conflict:  $a_{G_3, T_1} \circ_{G_5} a_{G_6, T_1}$ "  
return "conflict:  $a_{G_3, T_1} \circ_C a_{G_6, T_1}$ ".

## 6.2 Addition of new members to groups

Consider the addition of a member  $m$  to a group  $G_k$ . Inconsistencies may arise if subject  $m$ , any of its

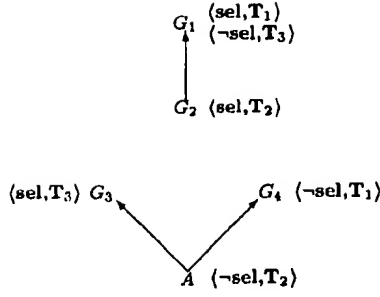


Figure 8. An authorization state

members, or any of the groups to which  $m$  itself or any of  $m$ 's members belong, owns some authorization which conflicts with an authorization owned by group  $G_k$  or by any of the groups to which  $G_k$  belongs.

For instance, consider the addition of group  $G_3$  as a member of group  $G_2$  in the authorization state shown in Figure 8. This operation would introduce the following conflicts:

- Authorization  $(sel, T_1)$  of  $G_1$  would conflict, with respect to user  $A$ , with authorization  $(-sel, T_1)$  owned by  $G_4$ . Formally:  $a_{G_1, T_1} \diamond_A a_{G_4, T_1}$ .
- Authorization  $(sel, T_2)$  of  $G_2$  would conflict, with respect to user  $A$ , with authorization  $(-sel, T_2)$  owned by  $A$ . Formally:  $a_{G_2, T_2} \diamond_A a_{A, T_2}$ .
- Authorization  $(-sel, T_3)$  of  $G_1$  would conflict, with respect to  $G_3$  and  $A$ , with authorization  $(sel, T_3)$  owned by  $G_3$ . Formally:  $a_{G_1, T_3} \diamond_{G_3} a_{G_3, T_3}$  and  $a_{G_1, T_3} \diamond_A a_{G_3, T_3}$ .

Figure 9 illustrates an algorithm for determining whether the addition of a member to a group would result in an inconsistent authorization state. The algorithm returns *ok* if the resulting authorization state is consistent; it returns the conflicts which would arise otherwise. Note that, again, for sake of simplicity, conflicts which are implied by other conflicts are not returned, that is, if there would be a conflict between two authorizations with respect to subject  $s$  only because of the membership of  $s$  to another subject  $s'$  with respect to which the two authorizations conflict, then only the conflict with respect to subject  $s'$  is returned.

The algorithm in Figure 9 works as follows. Consider the addition of member  $m$  to group  $G_k$ . In step 1, a check is made to determine whether  $m$  is already a member, either direct or indirect, of  $G_k$ .<sup>5</sup> If  $m$  is

<sup>5</sup>Note that a subject can belong to a group through different membership paths.

#### Check\_member( $m, G_k$ )

1. If  $\exists n \geq 1$  such that  $m \in_n G_k$  then return *ok*.
2.  $Anc := \{s \in G \mid \exists n \geq 0, G_k \in_n s \text{ and } \beta_i \geq 0, m \in_i s\}$
3.  $Disj\_Desc := \{s \in G \mid \exists n \geq 0, s \in_n m \text{ and } \beta_i \geq 0, s \in_i G_k\}$
4.  $Anc\_Disj\_Desc := \{s \in G - (Disj\_Desc \cup Anc) \mid \exists n \geq 1, s_i \in Disj\_Desc : s_i \in_n s\}$
5.  $Confl\_Desc := \{(a_i, a_j) \in AS \mid s(a_i) \in Disj\_Desc, s(a_j) \in Anc, p(a_i) = p(a_j), t(a_i) = t(a_j), pt(a_i) \neq pt(a_j), at(a_i) = at(a_j) = \text{strong}\}$
6.  $Confl\_AD := \{(a_i, a_j) \in AS \mid s(a_i) \in Anc\_Disj\_Desc, s(a_j) \in Anc, p(a_i) = p(a_j), t(a_i) = t(a_j), pt(a_i) \neq pt(a_j), at(a_i) = at(a_j) = \text{strong}\}$
7. If  $Confl\_Desc \cup Confl\_AD = \emptyset$  then return *ok* and exit
8. For each  $(a_i, a_j) \in Confl\_Desc$  do  
return "conflict:  $a_i \diamond_{s(a_i)} a_j$ " endfor
9. For each  $(a_i, a_j) \in Confl\_AD$  do  
   $Confl(a_i, a_j) := \{s \mid s \in_n m, s \in_n s(a_i)\}$   
   $Disj\_Confl(a_i, a_j) := \{s \in Confl(a_i, a_j) \mid \exists mp \in (MP(s, m) \cup MP(s, s(a_i))), \exists s', s' \neq s, s' \in Confl(a_i, a_j), s' \in mp\}$   
  For each  $s \in Disj\_Confl(a_i, a_j)$  do  
    return "conflict:  $a_i \diamond_s a_j$ "  
  endfor  
endfor.

Figure 9. Evaluation of the consistency of the authorization state resulting from addition of member  $m$  to group  $G_k$

already a member of  $G_k$ , then no inconsistencies can arise by the execution of the operation and *ok* is returned. Otherwise, the algorithm continues to execute. In step 2, set  $Anc$  of all the groups to which group  $G_k$  belongs (including  $G_k$  itself), and to which  $m$  does not belong, is determined. In step 3, set  $Disj\_Desc$  of all the subjects which belong to  $m$  (including  $m$  itself) and do not belong to  $G_k$  is determined. Then, in step 4, set  $Anc\_Disj\_Desc$  of all the groups (not already in  $Disj\_Desc$  or in  $Anc$ ) to which any of the subjects in  $Disj\_Desc$  belongs is determined. In step 5, set  $Confl\_Desc$  of all pairs of strong authorizations with the same privilege and table but with different privilege type such that the subject of one of them is in  $Disj\_Desc$  and the subject of the other is in  $Anc$  is determined. In step 6, set  $Confl\_AD$  of all pairs of strong authorizations with the same privilege and table but with different privilege type such that the subject of one of them is in  $Anc\_Disj\_Desc$  and the subject of the other one is in  $Anc$  is determined. If sets  $Confl\_Desc$  and  $Confl\_AD$  are empty, no pair of conflicting authorizations exists, and hence *ok* is returned (step 7). Otherwise, the conflicts which would arise are returned as

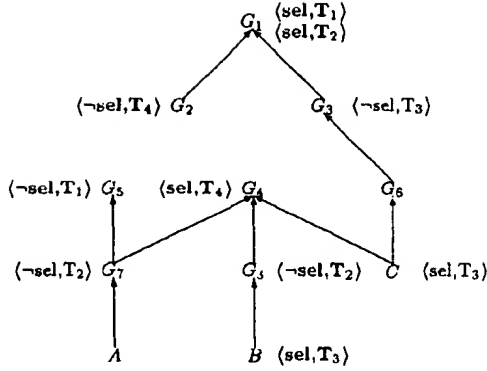


Figure 10. An authorization state

follows. In step 8, the conflict between every pair of authorizations  $\langle a_i, a_j \rangle$  in  $Confl\_Desc$  with respect to subject  $s(a_i)$ , which is a member of  $m$ , is returned. Finally, in step 9, every pair of authorizations  $\langle a_i, a_j \rangle$  in  $Confl\_AD$  is examined. Then, to avoid returning a conflict with respect to a subject which belongs to  $s(a_i)$  and  $m$  (and therefore would indirectly belong to  $s(a_j)$ ) only because of his membership of another subject for which the conflict is also returned, the following process is followed. For each pair  $\langle a_i, a_j \rangle \in Confl\_AD$ , set  $Confl(a_i, a_j)$  of all the subjects with respect to which  $a_i$  would conflict with  $a_j$ , i.e., all subjects which belong to both  $s(a_i)$  and  $m$  is determined. Then, set  $Disj-Confl(a_i, a_j)$  of all subjects in  $Confl(a_i, a_j)$  for which there exists a membership path to  $s(a_i)$  or to  $m$  which does not contain any other subject in  $Confl(a_i, a_j)$  is determined. For each of subject  $s \in Disj-Confl(a_i, a_j)$ , the conflict between authorizations  $a_i$  and  $a_j$  with respect to subject  $s$  is returned.

**Example 4** Consider the authorization state illustrated in Figure 10. Suppose a request is submitted to the system requiring the addition of group  $G_4$  as a member of group  $G_3$ . The execution of  $Check\_member(G_4, G_3)$  returns the following conflicts:  $a_{G_5, T_1} \circ_{G_3} a_{G_1, T_2}$  and  $a_{G_5, T_1} \circ_{G_7} a_{G_1, T_1}$ .  $\square$

## 7 Resolving conflicts among weak authorizations

Our model allows the simultaneous presence of conflicting weak authorizations. Hence, the authorization state may contain two weak authorizations, one stating that an access should be granted to some subject, and

the other one stating that the access should be denied to the subject. The way we solve this dilemma is by adopting a safe policy from a security viewpoint and denying access.

In this section, we show that using the default of denial is not the only option for resolving conflicts between weak authorizations. Users can resolve conflicts by inserting additional authorizations in the authorization state. This means that the users do not have to accept the system default option if they do not want; they can also eliminate conflicts in ways that best suit their needs. This is expressed by the following definition.

**Definition 7 (Resolution of conflicts)** A conflict between two weak authorizations  $a_i$  and  $a_j$  with respect to subject  $s$  has a resolution iff it is possible to specify an authorization overriding one of these (with respect to subject  $s$ ).

For example, consider the authorization state shown in Figure 5. The authorizations specified for groups  $G_4$  and  $G_5$  conflict over user  $B$ . The conflict can be resolved by specifying an additional authorization, either positive or negative, for  $B$ .

We claim that every conflict, with one exception, can be resolved by the addition of a new authorization. The exception occurs when conflicting authorizations are specified for the same subject.

To see this, suppose that there exist two authorizations  $a_i$  and  $a_j$  for a privilege on a table that conflict with respect to a subject  $s$ . Thus,  $p(a_i) = p(a_j)$  and  $t(a_i) = t(a_j)$ , but  $pt(a_i) \neq pt(a_j)$ . There are three cases to consider.

**Case 1:**  $s(a_i) \neq s(a_j)$ .

Clearly,  $s \neq s(a_i)$  and  $s \neq s(a_j)$  (since otherwise  $a_i$  and  $a_j$  will not conflict with respect to  $s$ ). The conflict between  $a_i$  and  $a_j$  with respect to subject  $s$  can be resolved by inserting another authorization  $a_m$  with  $p(a_m) = p(a_i)$ ,  $t(a_m) = t(a_i)$ ,  $s(a_m) = s$ , and either  $pt(a_m) = +$  or  $pt(a_m) = -$ , depending on whether the grantor wishes to give or deny the privilege to  $s$ .

**Case 2:**  $s(a_i) = s(a_j)$ , but  $s \neq s(a_i)$ .

The conflict can once again be resolved by inserting another authorization  $a_m$  as given in case 1 above.

**Case 3:**  $s(a_i) = s(a_j) = s$

In this case, the conflict can only be solved by either deleting one of the two authorizations or inserting a strong authorization  $a$  with  $s$  as subject and same privilege and table as the two conflicting authorizations. Note, however, that the insertion of the strong authorization may not always be possible, since it may result in an inconsistent authorization state. We claim that it is proper in this case to require that one of the

two authorizations be deleted in order to resolve the conflict since the simultaneous presence of a negative and a positive authorization specified for  $G_1$  is intrinsically ambiguous.

Note that conflicts between weak authorizations may arise when any of the administrative operations are executed. In particular, operations that decrease the authorizations of subjects, such as removal of users from groups and revocation of authorizations, may generate new conflicts.

## 8 The mediator

In this section, we illustrate how the mediator is used to mirror different protection policies. In particular, we show that the following policies can be expressed in our system: the traditional closed and open policies, the closed policy with negation and the enforcement of the denials-take-precedence principle, and the closed policy with negation and the enforcement of the most specific authorization takes precedence principle. Note that the last policy is the one actually enforced by the model implemented by our mechanism. The first three policies can be supported by using only some of the features that our model provides.

The implementation of a classical closed policy is straightforward in our model. It is the task of the mediator to ensure that only positive authorizations can be specified; the mediator rejects any attempts by the users to specify negative authorizations.

For the open policy, the mediator must ensure that the following conditions exist: For every object, a weak positive authorization is specified for the group representing the root of the graph.<sup>6</sup> Users will then be allowed to specify only strong negative authorizations. Since a strong negative authorization overrides the weak positive authorization in our model, the result is an open policy where all accesses for which users do not specify negative authorizations are allowed. Note that current DBMSs have a group, called **public**, containing all users of the system, to which authorizations can be granted; however, since negative authorizations are not supported, no exceptions to the authorizations granted to the group **public** can be enforced.

The denials-take-precedence policy can be implemented by the mediator by ensuring that all positive authorizations are weak and all negative authorizations are strong so that negative authorizations will always override positive authorizations.

As stated earlier, a major advantage of using our framework is that all users are not constrained to the

<sup>6</sup>If the membership graph is not single rooted, a group, called **public**, containing all subjects in the system can be inserted.

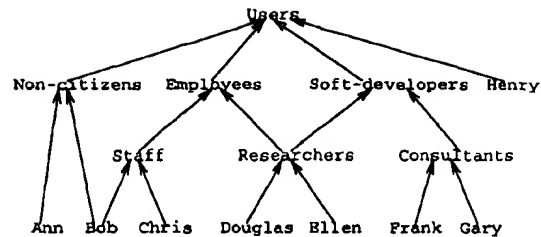


Figure 11. An example of group membership graph

use of a single policy. They can choose to apply different policies on different relations, as shown in the following example. Suppose the group membership graph is as given in Figure 11 and suppose Henry creates the following tables with their different protection requirements:

- **Public.info:** Everybody is to be allowed access. This can be accomplished by granting a strong positive authorization to **Users**.
- **Nat.pub.info:** Everybody who is a citizen has access. A weak positive authorization can be granted to **Users** and a strong negative authorization granted to **Non-citizens**.
- **Reports:** Everybody can access unless explicitly denied. A weak positive authorization is granted to **Users**. Negations can be specified at any time.
- **Organization.info:** Henry wishes to retain control of all access authorizations on the relation except for the **select** privilege which can be administered by the members of the group **Staff**. Moreover, Henry does not want Gary to be able to read information in the relation. To this end, Henry grants the weak **adm-access** for the **select** privilege on the relation to **Staff** and grants a negative strong authorization for the **select** privilege on the relation to Gary.
- **Projects:** Projects contains information about software projects. People involved in software development should be authorized. A weak positive authorization is specified for **Soft-developers**.
- **Projects:** Projects temporarily contains some private information that is not to be released for some time. Consultants must be temporarily forbidden access to the relation. A negative (strong

or weak) authorization can be specified for the select privilege on the relation for Consultants. There is no need to revoke the authorization previously granted since it will once again become valid upon revocation of the denial. □

## 9 Related Work

Early authorization models [8] were based on the closed world policy and accordingly only allowed the specification of positive authorizations. More recent authorization models also permit specification of negative authorizations stating accesses to be denied. In some of these models, conflicts are solved simply by adopting the denials-take-precedence policy [4, 17, 14, 21]. Other models provide more sophisticated conflict resolution policies [18, 19, 7, 22].

The concept of strong and weak authorizations adopted in our model has been first introduced in the authorization model of Orion [19]. In this model, authorizations can be specified only for groups of users (called *roles* in Orion), not for single users. Positive authorizations granted to a group propagate to all the members of the group. Negative authorizations granted to a group propagate to all the groups to which it belongs. Resolution of conflicts is based on the concept of more specific authorization. The Orion authorization model has the merit of having introduced the concept of strong and weak authorizations. However, it suffers from several limitations, which we have tried to overcome in our model. First, in the Orion authorization model, negative authorizations propagate from a subject to the groups to which the subject belongs (not vice versa like in our model). This means that it is not possible to grant a negative authorization to a group which holds for all the members of the group. Second, in the Orion authorization model, authorizations cannot be granted to single users. Moreover, users are not taken into consideration when evaluating the consistency of the authorization state; a user can belong to groups holding conflicting authorizations. Since a user is permitted an access if he belongs to a group that has a positive authorization for it, in this case, access is authorized. Then, it is not possible to really enforce strong authorizations at the level of single users. Third, the Orion authorization model requires the authorization state to be complete, i.e., for every possible access an authorization, either positive or negative, must exist. This approach forces the use of negative authorizations to represent cases where the authorization is simply not to be given and makes the authorization administration task very difficult. Fourth, the Orion authorization model does not provide any administra-

tive policy: every user authorized for an access can also grant authorizations for that access. This approach raises several problems. In particular, it is impossible for the owner of an object to retain control of the users that can access his objects.

With respect to multipolicy models, Jonscher and Dittrich [17] present an access control system for the protection of information in distributed federated database systems which allow the enforcement of different policies. Each policy is characterized by 19 attributes referring to different policy aspects (e.g., types of privileges allowed, signs of authorizations allowed, subjects' hierarchies to be considered). Different reference monitors are then defined, each enforcing a particular policy. The behavior of each access monitor is determined by the attributes specified for the policy implemented by the monitor. Each access request submitted to the system is forwarded to the responsible reference monitor for the object to be accessed and allowed or denied accordingly.

Other work on multipolicy aspects concerns the integration and coexistence of different, possibly inconsistent, policies [10]. Indeed, where different systems applying different policies interact, the problem of which policy to apply with respect to the common process arises. Hosmer [16] describes a multipolicy paradigm for supporting the enforcement of different protection policies. The approach is based on the concept of metapolicy that is a policy about policies. A conceptual model of this "multipolicy machine" has been proposed by Hell [2]. The model provides a formal framework for dealing with the combination of unspecified policies.

## 10 Conclusions

The usefulness of separating security policies from security mechanisms in the development of access control systems has long been recognized [12, 9]. Security policies are high-level guidelines specifying how access is to be controlled. Mechanisms are sets of functions implementing protection policies. Among the advantages of this separation is that it is possible to change the policy without requiring changes to the underlying mechanism [9]. In spite of the policy-mechanism separation principle, access control systems today are based on a mechanisms enforcing a specific set of predefined policies.

In this paper, we have moved a step toward the development of a flexible access control mechanism that can support different policies. In particular, we have presented an authorization model on which such a flexible mechanism can be based. The model permits both

positive and negative authorizations, permits authorizations that must be strongly obeyed and authorizations that allow for exceptions, and enforces ownership together with delegation of administrative privileges. The flexibility of the model makes it possible to represent different policies. We have presented the model and illustrated how it can be used to express different policies and different protection requirements.

Our proposal represents only an initial step in the development of multipolicy mechanisms and much work still needs to be done. Issues that we plan to investigate include the identification of different policies with respect to which users can tailor the system; the definition of the mapping functions between the policies and the reference authorization model; the ability to support unspecified policies; the regulation of the co-existence of multiple, possibly conflicting, policies on the same object; and the efficient implementation of the mechanism.

## References

- [1] R. Baldwin. Naming and grouping privileges to simplify security management in large databases. In *Proc. IEEE Symp. on Security and Privacy*, pages 61-70, Oakland, CA, 1990.
- [2] D. Bell. Modeling the "Multipolicy Machine". In *Proc. of the New Security Paradigm Workshop*, pages 2-9, Little Compton, RI, USA, 1994.
- [3] E. Bertino, S. Jajodia, and P. Samarati. Access controls in object-oriented database systems: Some approaches and issues. In N. Adam and B. Bhargava, editors, *Advanced Database Concepts and Research Issues*, pages 17-44. Springer-Verlag LNCS 759, 1993.
- [4] E. Bertino, P. Samarati, and S. Jajodia. An extended authorization model for relational databases. *IEEE TKDE*. To appear.
- [5] E. Bertino, P. Samarati, and S. Jajodia. Authorizations in relational database management systems. In *Proc. ACM Conf. on Comp. and Comm. Security*, pages 130-139, Fairfax, VA, 1993.
- [6] P. Bonatti, S. Kraus, and V. Subrahmanian. Declarative foundations of secure deductive databases. In *Proc. Int'l. Conf. on Database Theory*. Springer-Verlag LNCS 303, May 1992.
- [7] H. Drüggenmann. Rights in an object-oriented environment. In C. Landwehr and S. Jajodia, editors, *Database Security, V: Status and Prospects*. North-Holland, 1992.
- [8] S. Castano, M. Fugini, G. Martella, and P. Samarati. *Database Security*. Addison-Wesley, 1994.
- [9] D. Denning. *Cryptography and Data Security*. Addison Wesley, 1982.
- [10] J. Dobson and J. McDermid. A framework for expressing models of security policy. In *Proc. IEEE Symp. on Security and Privacy*, pages 229-239, Oakland, CA, May 1989.
- [11] R. Fagin. On an authorization mechanism. *ACM TODS*, 3(3):310-319, Sept. 1978.
- [12] E. Fernandez, R. Summers, and C. Woods. *Database Security and Integrity*. Addison Wesley, 1981.
- [13] R. Gagliardi, G. Lapis, and B. Lindsay. A flexible and efficient database authorization facility. Technical Report RJ 6826(65360), IBM Research Division, Almaden Research Center, November 1989.
- [14] N. Gal-Oz, E. Gudes, and E. Fernandez. A model of methods authorization in object-oriented databases. In *Proc. VLDB*, pages 52-61, Dublin, Ireland, 1993.
- [15] P. Griffiths and B. Wade. An authorization mechanism for a relational database system. *ACM TODS*, 1(3):243-255, Sept. 1976.
- [16] H. Hosmer. Multipolicy paradigm II. In *Proc. of the New Security Paradigm Workshop*, Little Compton, RI, 1992.
- [17] D. Jonscher and K. Dittrich. Argos - A configurable access control system for interoperable environments. In *Proc. of the IFIP WG11.3 Working Conference on Database Security*, pages 39-66, Rensselaerville, NY, 1994.
- [18] T. Lunt. Access control policies for database systems. In C. Landwehr, editor, *Database Security II: Status and Prospects*, pages 41-52. North-Holland, 1989.
- [19] F. Rabitti, E. Bertino, W. Kim, and D. Woelk. A model of authorization for next-generation database systems. *ACM TODS*, 16(1):89-131, March 1991.
- [20] R. Sandhu and P. Samarati. Access control: Principles and practice. *IEEE Comm.*, pages 2-10, Sept. 1994.
- [21] M. Satyanarayanan. Integrating security in a large distributed system. *ACM TOCS*, 7(3):247-280, August 1989.
- [22] H. Shen and P. Dewan. Access control for collaborative environments. In *Proc. Int'l. Conf. on Computer Supported Cooperative Work*, pages 51-58, November 1992.
- [23] I. Software. *Informix-OnLine/Secure Security Features User's Guide*. Inc., Menlo Park, CA, 1993.
- [24] G. Wiederhold. Mediators in the architecture of future information systems: A new approach. *IEEE Computer*, pages 38-49, 1992.
- [25] P. Wilms and B. Lindsay. A database authorization mechanism supporting individual and group authorizations. In R. van de Riet and W. Litwin, editors, *Distributed Data Sharing Systems*, pages 273-292. North-Holland, 1982.